PERCEPTRÓN MULTICAPA CLASIFICACIÓN

Práctica 2

Introducción a los Modelos Computacionales
Grado en Ingeniería Informática
Universidad de Córdoba – Escuela Politécnica Superior

Índice

Descripción del modelo neuronal

El objetivo de esta práctica es que el perceptrón multicapa desarrollado en la anterior práctica consiga resolver problemas de clasificación. Para ello, se les dará a las salidas un significado probabilístico, es decir, mostrarán la probabilidad de que el resultado pertenezca a una clase u otra. La neurona que tenga la salida con el valor más alto será la clase a la que pertenece dicho patrón que ha predicho la red neuronal.

Para que los resultados obtengan un significado probabilístico se usarán dos elementos:

- Uso de una función *softmax* en la capa de salida.
- Uso de la función de error basada en entropía cruzada.

Para ello se ha añadido la posibilidad de que la capa de salida sea una función *softmax* en la función que aplica el algoritmo de retropopagación del error y la que propaga las entradas. Al haber introducido la función de error basada en entropía cruzada, se ha tenido que modificar tanto la función de retropropagación del error como la función que calcula el error de salida.

Para realizar estos cambios se han tenido en cuenta las 4 posibles combinaciones de función de error y función en la capa de salida:

- *Sigmoide* y función de error *MSE*
- *Sigmoide* y función de error *Entropía Cruzada* (sin sentido práctico)
- *Softmax* y función de error *MSE*
- *Softmax* y función de error *Entropía Cruzada*

También se han transformado los problemas a una representación 1-de-k, donde k es el número de clases. Por cada patrón habrá un vector de k elementos, donde el elemento i-ésimo será 1 si el patrón pertenece a esa clase y 0 si no pertenece para los valores esperados, que indican que un patrón tiene probabilidad 1 de pertenecer a una clase. Para los valores estimados, la distribución es la misma, pero lo valores estarán comprendidos entre 0 y 1, indicando la probabilidad de pertenecer a cada una de las clases.

Una vez controlado esto, se ha introducido en el algoritmo su versión *offline*, es decir, que el ajuste de los pesos de la red neuronal se realice una vez han pasado todos los patrones en lugar de que se ajusten a cada patrón que pasa. Para implementar esta versión simplemente había que realizar unos cambios en las funciones que se encargan de simular la red. Para esto también hay que tener en cuenta que en la función que se encarga de ajustar los pesos hay que dividir los nuevos valores para los pesos entre el número de patrones usados.

También se ha incorporado el cálculo del *CCR* (Correctly Classified Ratio) o porcentaje de patrones correctamente clasificados como función de rendimiento, que nos dará una idea de lo buena que es la red con las diferentes configuraciones. A mayor *CCR* mejor será la configuración de la red, pues significa que clasifica bien una gran cantidad de patrones.

Pseudocódigos

En este apartado se muestran los pseudocódigos de las funciones más relevantes del código implementado. Las funciones seleccionadas han sido las que se encargan de simular la red online, alimentar las entradas, propagar esas entradas, retropropagar el error cometido en la estimación, acumular el cambio que se le va a aplicar a los pesos y ajustar los pesos.

Entrenar

```
Si es offline
        Para 1 (i) hasta número de capas
                Para 1 (j) hasta número de neuronas de la capa i
                        Para 1 (k) hasta número de neuronas de la capa i-1
                                Δw^i_jk(t − 1) ← Δw^i_jk
                                Δw^i_jk ← 0
                        finPara
                finPara
        finPara
finSi
Para 1(i) hasta número de patrones
        simularRed(entradas, salidas, funcionError)
finPara
Si es offline
        ajustarPesos()
finSi
```

Simular Red

```
Si es online
        Para 1 (i) hasta número de capas
                Para 1 (j) hasta número de neuronas de la capa i
                        Para 1 (k) hasta número de neuronas de la capa i-1
                                Δw^i_jk(t − 1) ← Δw^i_jk
                                Δw^i_jk ← 0
                        finPara
                finPara
        finPara
finSi
alimentarEntradas(entrada)
propagarEntradas()
retropropagarError(objetivo)
acumularCambio()
Si es online
        ajustarPesos()
finSi
```

Propagar Entradas

Para 1 (i) hasta número de capas (menos la salida)
    Para 1 (j) hasta número de neuronas de la capa i
        Si sesgo activo
        $net \leftarrow sesgo$
        Si no
        $net \leftarrow 0$
        Para 1 (k) hasta número de neuronas de la capa i-1
            $net_j^i \leftarrow net_j^i + x_k^i * w_{jk}^i$
        finPara
        $x_j^i \leftarrow \frac{1}{1 + e^{-net_j^i}}$
    finPara
finPara

//Para sigmoide
Para 1 (j) hasta número de neuronas de la capa de salida (i)
    Si sesgo activo
    $net \leftarrow sesgo$
    Si no
    $net \leftarrow 0$
    Para 1 (k) hasta número de neuronas de la capa i-1
        $net_j^i \leftarrow net_j^i + x_k^i * w_{jk}^i$
    finPara
    $x_j^i \leftarrow \frac{1}{1 + e^{-net_j^i}}$
finPara

//Para softmax

Para 1 (j) hasta número de neuronas de la capa de salida (i)
    Si sesgo activo
    $net \leftarrow sesgo$
    Si no
    $net \leftarrow 0$
    Para 1 (k) hasta número de neuronas de la capa i-1
        $net_j^i \leftarrow net_j^i + x_k^i * w_{jk}^i$
    finPara
    $sumNet \leftarrow sumNet + e^{-net_j^i}$
    $sumV \leftarrow e^{-net_j^i}$
finPara

Para 1 (j) hasta número de neuronas de la capa de salida (i)
    $x_j^i \leftarrow \frac{sumV(j)}{sumNet}$
finPara

Retropropagar Error

Si uso sigmoide
Para 1 (i) hasta número de neuronas de la capa de salida
Si uso entropía cruzada
$\delta_i^n \leftarrow -(\frac{objetivo[i]}{x_i^n}) * x_i^n * (1 - x_i^n)$ (n es el nº de neuronas de la última capa)
Si uso MSE
$\delta_i^n \leftarrow -(objetivo[i] - x_i^n) * x_i^n * (1 - x_i^n)$ (n es el nº de neuronas de la última capa)
finSi
finPara

Si uso softmax
Para 1 (i) hasta número de neuronas de la capa de salida
$sumaSoft \leftarrow 0$
Para (j) hasta número de neuronas de la capa de salida
Si uso entropía cruzada
$parte1 \leftarrow \frac{objetivo[j]}{x_i^n}$
Si uso MSE
$parte1 \leftarrow objetivo[j] - x_i^n$
finSi
$parte2 \leftarrow x_i^n * (I(i == j) - x_i^n)$
$sumaSoft \leftarrow parte1 * parte2$
finPara
$\delta_i^n \leftarrow sumaSoft$
FinPara
finSi

Para número de capas -2 (i) hasta 0 (decremento)
Para 1 (j) hasta número de neuronas de la capa i
//Pesos
Para 1 (k) hasta número de neuronas de la capa i+1
$\delta_j^i \leftarrow \delta_j^i + \Delta w_{kj}^{i+1} * \delta_j^{i+1}$
finPara
//Sesgo
Para 1 (k) hasta número de neuronas de la capa i+1
$\delta_j^i \leftarrow \delta_j^i + \Delta w_{kn}^{i+1} * \delta_j^{i+1}$ (n es el nº de neuronas de esta capa)
finPara
$\delta_j^i \leftarrow \delta_j^i * x_j^i * (1 - x_j^i)$
finPara
finPara

Ajustar pesos

Para 1 (i) hasta número de capas
Para 1 (j) hasta número de neuronas de la capa i
Para 1 (k) hasta número de neuronas de la capa i-1
$w_{jk}^i \leftarrow \frac{w_{jk}^i - \eta * \Delta w_{jk}^i - \mu * \left(\eta * \Delta w_{jk}^i (t-1)\right)}{nNumPatronesTrain}$
finPara
$w_{jn}^i \leftarrow \frac{[\![w]\!]_{jn}^i - \eta * \Delta w_{jn}^i - \mu * \left(\eta * \Delta w_{jn}^i (t-1)\right)\Big)}{nNumPatronesTrain}$ (n es el nº de neuronas de la capa i-1)
finPara
finPara

Experimentos y análisis de los resultados

Descripción de las bases de datos

Todos los ficheros que contienen las bases de datos tienen el mismo formato:

- En la primera línea se encuentran 3 enteros que indican el número total de entradas del problema(*n*), el número total de salidas del problema(*k*) y el número total de patrones en dicho fichero(*p*).

- A partir de la segunda línea, cada línea corresponde a un patrón:
    - Los primeros *n* valores corresponden a las entradas.
    - Los siguientes *k* valores son las salidas deseadas.

Descripción de los parámetros considerados

Los parámetros que se han considerado en esta práctica para el estudio del comportamiento del perceptrón son los siguientes:

- Nº de iteraciones: Es un entero que indica el número máximo de iteraciones que se van a realizar en el bucle externo. Por cada iteración se le pasarán todos los patrones.
- Nº de capas ocultas: Es un entero que indica el número de capas ocultas de la red neuronal. Mientras mayor sea el número, más compleja será la red.
- Nº de neuronas en capa oculta: Es un entero que indica el número de neuronas que tendrá cada una de las capas ocultas. Mientras mayor sea el número, más compleja será la red.
- Valor de *eta*(η): Es un valor real entre 0 y 1 que indica el factor de aprendizaje del modelo.
- Valor de *mu*(μ): Es un valor real entre 0 y 1 que indica el factor de momento del modelo.
- Sesgo: Es un booleano que indica si el modelo tendrá o no sesgo.
- Versión: Puede ser *on-line* u *off-line.* Es un booleano que indica cuál se usa.
- Función de error: La función de error que se utiliza durante el aprendizaje. MSE (0) o entropía cruzada (1).
- Función en capa de salida: Se podrá utilizar o *softmax* o la *sigmoide*. Se indica con un booleano.

Resultados obtenidos

En este apartado se explicarán los diferentes experimentos que se han realizado con el programa desarrollado y sus resultados, con el fin de estimar la configuración de los parámetros óptima para cada uno de los problemas, es decir, encontrar la serie de parámetros con la que obtendremos los mejores resultados posibles. Se harán experimentos con la arquitectura de la red neuronal utilizada, con la activación y desactivación del sesgo y del momento y con el cambio en el factor de aprendizaje. Finalmente se mostrará la configuración con mejores resultados que será la que se analizará en el apartado siguiente.

Comparación de arquitecturas

	Error				CCR			
	Entrenamiento		Test		Entrenamiento		Test	
Estructura	Media	Desv	Media	Desv	Media	Desv	Media	Desv
{n : 5 : k}	0,04751	0,0019226	0,07024	0,003741	87,033	1,77036	77,4295	2,69665
{n : 10 : k}	0,02196	0,0026012	0,04841	0,004383	95,5259	0,940279	85,3918	1,37717
{n : 50 : k}	0,00617	0,0002205	0,03438	0,003156	99,7645	0,184082	88,7147	1,10832
{n : 75 : k}	0,00423	0,000364	0,0337	0,003078	99,8901	0,0894957	89,7806	0,903126
{n : 5 : 5 : k}	0,08547	0,0121383	0,10329	0,007787	72,5903	5,51225	64,5768	5,38419
{n : 10 : 10 : k}	0,03122	0,0037318	0,05964	0,004398	92,3862	2,09886	82,069	1,69395
{n : 50 : 50 : k}	0,00518	0,0007381	0,03525	0,002123	99,7017	0,102342	89,3417	0,586467
{n : 75 : 75 : k}	0,00306	0,0002598	0,03596	0,004264	99,9686	0,0429924	88,5893	1,75381

Los resultados son bastante similares entre las redes con una capa oculta y con dos capas ocultas, en ambos aumenta la precisión a la vez que se va aumentando el número de neuronas en las capas ocultas. El mejor resultado es una sola capa oculta con 75 neuronas. Esto puede deberse a que el modelo es más flexible y sobrentrena menos que con dos capas ocultas. Si el problema es más complejo suele necesitar más capas, pero, en general, cuanto más simple sea el modelo, mejor.

Combinaciones de funciones de error y funciones de activación

XOR

	Error				CCR			
	Entrenamiento		Test		Entrenamiento		Test	
Combinación	Media	Desv	Media	Desv	Media	Desv	Media	Desv
MSE-Sigmoide	0,00407314	0,00061703	0,00406968	0,00061729	100	0	100	0
MSE-Softmax	0,0508976	0,111457	0,0508933	0,111458	95	11,1803	95	11,1803
Entropía Cruzada-Softmax	0,00137837	0,00014548	0,00137279	0,00014294	100	0	100	0

Para el problema “xor” los mejores resultados son las combinaciones MSE-*Sigmoide* y Entropía Cruzada-*Sigmoide*, ya que la función de error MSE no es la función natural de error cuando se tienen salidas probabilísticas (función *Softmax*) porque trata por igual cualquier diferencia de error. La primera combinación es muy buena porque con la función de error MSE aproxima los valores de las salidas a sólo 2 valores, 0 y 1, que son las dos posibles salidas del problema “xor”.

DIGITS

	Error				CCR			
	Entrenamiento		Test		Entrenamiento		Test	
Combinación	Media	Desv	Media	Desv	Media	Desv	Media	Desv
MSE-Sigmoide	0,0398228	0,0114158	0,0451261	0,00992816	73,3909	11,2717	70,0313	9,76478
MSE-Softmax	0,0250866	0,0149774	0,0344055	0,0147799	84,0345	11,6352	75,9248	10,6061
Entropía Cruzada-Softmax	0,00423263	0,00036397	0,0337049	0,00307762	99,8901	0,0894957	89,7806	0,903126

En este caso, la mejor es la última combinación, ya que este problema es multiclase. Por esto, la función de error *Sigmoide* no funciona correctamente, ya que está pensada para problemas biclase. La combinación de MSE-*Softmax* no es buena tampoco por lo que se explicó anteriormente.

ENTROPÍA CRUZADA Y SIGMOIDE

Esta combinación no se realiza porque son incompatibles dichas funciones. La función de activación *sigmoide* asigna el valor de la salida a una de las clases. Si el valor es próximo a 0, asignará a la clase 0, si el valor es cercano a 1, lo asignará a la clase 1. La función *softmax* fuerza que la suma de las salidas sea 1, haciendo que las salidas se transformen en probabilidades, por lo que hay más diferencia entre las salidas. Esto favorece la labor de la función de entropía cruzada, que es potenciar la salida más alta. Sin embargo, con la función *sigmoide*, los valores pueden salir muy próximos y no haber tanta diferencia entre ellos, ya que su suma no está forzada a que valga 1, por lo que la entropía cruzada no trabajará bien.

Comparación *on-line – off-line*

XOR

	Error				CCR			
	Entrenamiento		Test		Entrenamiento		Test	
Versión	Media	Desv	Media	Desv	Media	Desv	Media	Desv
off-line	0,00407314	0,000617028	0,00406968	0,000617287	100	0	100	0
on-line	0,000641593	4,94E-05	0,000637185	4,91E-05	100	0	100	0

En el caso del "xor" ambas versiones consiguen el 100% de CCR tanto en entrenamiento como en test, sin embargo, el error cometido es mucho menor en la versión on-line, ya que este para este problema los patrones de por sí no tienen significado en sí, si no que el conjunto de los cuatro patrones es el que muestra el problema completo.

DIGITS

	Error				CCR			
	Entrenamiento		Test		Entrenamiento		Test	
Versión	Media	Desv	Media	Desv	Media	Desv	Media	Desv
off-line	0,00423263	0,000363965	0,0337049	0,00307762	99,8901	0,0894957	89,7806	0,903126
on-line (*eta* alto)	0,482369	0,155381	0,507871	0,160755	31,9309	13,8863	30,5329	13,4497
on-line (*eta* bajo)	2,53E-05	4,33E-06	0,0417492	0,00615767	100	0	91,2853	1,3556

Para el problema "digits" sí importa la versión que se utilice, ya que ahora cada patrón tiene significado por sí mismo, es decir, se puede decir que un patrón es un número determinado viéndolo. Si se actualiza patrón a patrón el modelo aprenderá a distinguir uno a uno, mientras que si se actualiza una vez han pasado todos, no va a aprender a distinguir cuál es cual, ya que no tienen significado en conjunto.

Sin embargo, si se reduce el valor de *eta* (factor de aprendizaje), se logra que el aprendizaje sea más lento, es decir, que no busque tan rápido el óptimo global. El factor de momento se mantiene al máximo porque sí se busca que se tenga en cuenta la trayectoria que lleva en la búsqueda, por si acaso hay que salir de un óptimo local. Al no ir tan rápido en la búsqueda se consigue que se tome todo el problema como un todo, mientras que si se va más rápido en la versión *off-line* (*eta* alto) se busca que aprenda lo más rápido posible un tipo de patrón (números del 0 al 9). Se llega a conseguir un modelo más preciso que en la versión *off-line*, pero el coste computacional es mayor, al igual que el coste temporal, por lo que se tomará la versión *on-line* como mejor configuración.

Mejor configuración

XOR

- Nº de iteraciones: 1000
- Nº de capas ocultas: 2
- Nº de neuronas en capa oculta: 100
- Valor de *eta*(η): 0.1
- Valor de *mu*(μ): 0.9
- Sesgo: Activado
- Funciones: MSE-*Sigmoide*
- Versión: *On-line*

DIGITS

- Nº de iteraciones: 300
- Nº de capas ocultas: 1
- Nº de neuronas en capa oculta: 75
- Valor de *eta*(η): 0.7
- Valor de *mu*(μ): 1
- Sesgo: Activado
- Funciones: Entropía Cruzada-*Softmax*
- Versión: *Off-line*

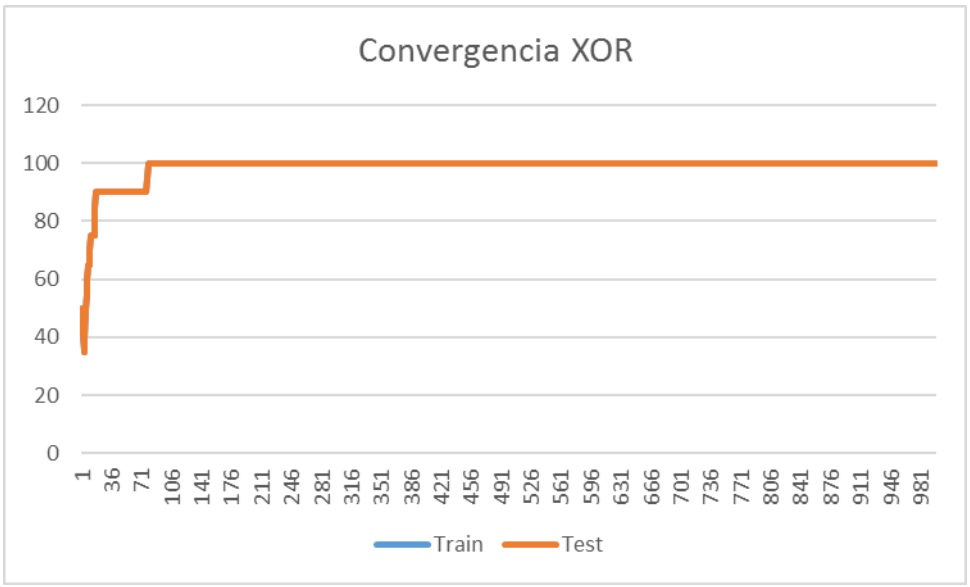
Análisis de los resultados

En este último apartado se mostrarán las gráficas de convergencia del perceptrón multicapa para cada uno de los problemas con los valores de los parámetros acordados anteriormente. En las gráficas se representarán en el eje de abcisas el número de iteraciones y en el de ordenadas el error cometido en la estimación. El CCR del entrenamiento se representa con la línea azul y el del test se representa por una línea naranja.

Gráficas de convergencia

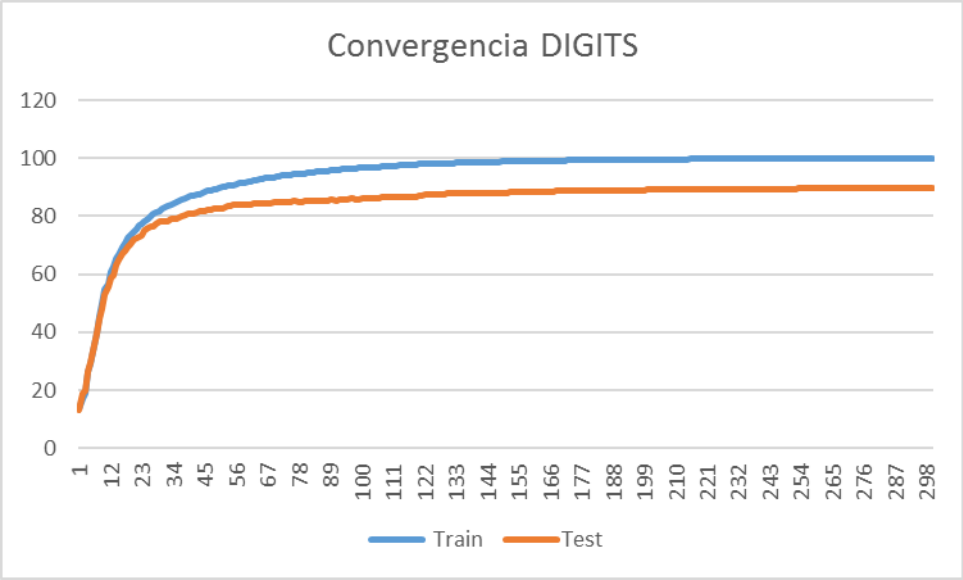
Se presentan las gráficas de convergencia para ambos problemas. Se han realizado con las mejores configuraciones de ambos.

XOR



En la gráfica se puede observar que con unas 100 iteraciones es suficiente para conseguir un CCR del 100%. Esto puede resultar contradictorio con las conclusiones a las que se llegaron en la práctica anterior, pero nada más lejos de la realidad. En la práctica anterior, las gráficas de convergencia se realizaron con el error cometido (MSE) en los conjuntos de entrenamiento y test a lo largo de las iteraciones. En este caso esto no se tiene en cuenta, sólo se muestra el CCR, por lo que si lo que se busca es que el CCR sea perfecto, con 100 iteraciones bastaría. Esto no quiere decir que el error cometido en la iteración 100 y en la iteración 1000 sea el mismo, que, como se vio en la práctica anterior, no lo es.

DIGITS



En este caso el modelo sobrentrena un poco, aunque siempre sigue aumentando el CCR, tanto en test como en train, por lo que las 300 iteraciones son necesarias para conseguir un buen CCR. Como se puede observar, el modelo llega al 90% de CCR de forma asintótica, por lo que poner más iteraciones no serviría de mucho. El problema es bastante más complejo que el anterior, lo que podría explicar que no llegue al 100% del CCR. También puede ser que modificando el factor de aprendizaje se pueda conseguir el 100%. Habría que aumentar el factor de aprendizaje, pues con el actual no hay ninguna irregularidad en las gráficas, así que si se pone valor más bajo se obtendrá un CCR peor.

Bibliografía

- Apuntes, transparencias y material referente a esta práctica de la asignatura en la plataforma Moodle.